IN THE CLAIMS:

Please amend the claims as follows:

Claim 1. (Currently withdrawn)

- A method enabling: (i) separation of application software in the application scenario layer, which describes interactions, rules, conditions and service calls, and the service layer, which describes services (ii) application layer description that reflects application requirements as business rules and scenarios presented as a sequence of scenario acts written by subject matter experts with close to natural language in business domain terms; (iii) interpretation of scenarios and business rules into interactions with the semantic-enabled component, like knowledgebase filled with business domain ontology, presentation components, and the underlying application services (iv) creation and modification of business rules and scenarios that comprise the application scenario layer at run-time (v) invocation of services designed as integration-ready components

comprising:

- receiving a sequence of application scenario acts with the following interpretation and execution of application flow where in each scenario act:

- receiving from computer programs or users, further called agents, application scenario instructions expressed with close to natural language business domain terms, where instructions can include but not limited to: prompts to interacting agents; descriptions of expected input messages,

further called events; business rules pointing to alternative set of scenario acts or scenarios to handle the events; descriptions of application services to be invoked based on business rules; presentation instructions

- interpreting descriptions of application services expressed with business domain terms into application service names and related variable names
- interpreting business rules expressed with business domain terms into Boolean logics which represent conditions for service invocations
- translating a user or a computer program input according to event handling instructions into application run-time variable values
- replacing business rules and scenario variable names with application run-time variable values
- checking conditions of the scenario acts by using resolved Boolean logics and run-time variable values
- selecting and invoking alternative application services based on checked conditions derived from business rules and scenarios
- translating service execution results into presentation format according to presentation instructions
- performing presentations over voice, screen or direct computer-to-computer interaction according to presentation instructions

Claim 2 (Currently withdrawn). The method of claim 1 further comprising:

- receiving application scenarios written with the Application Scenario Language that includes service and scenario invocations with rules and

conditions as well as expected events and presentation instructions expressed with business domain ontology terms;

- using semantic-enabled interpretation to resolve said scenarios, rules and instructions based on business domain terms, into Boolean conditions and traditional application and presentation service invocations and enabling developers or business experts to quickly build the application scenario layer by describing application flow as a set of scenarios that define interactions between system components and agents, where said agents can be users, human beings, or programs, for example, a partner program engaged in a common business transaction;
- consulting with similar systems which might have different knowledge resources to resolve scenario instructions which cannot be successfully interpreted by the system
- responding to similar systems which can send an unresolved scenario for a resolution;
- translating snapshots of scenario acts and scenarios into source code of a traditional fixed-form application with better performance and less flexibility;
- analysis of scenario interpretation and execution including: i) history of successes of interpretation and execution of scenarios, ii) history of interpretation failures, iii) learning scenarios that prompt an agent which can be a user or a program, to re-define the input or to provide more

details for interpretation, iv) queue of scenarios with un-answered prompts or questions to resolve unsuccessful interpretations;

- communicating to similar systems connected over the networks results of analysis of scenario interpretation and execution

Claim 3 (Currently withdrawn). The method of claim 2 further comprising application scenario acts that can define: (a) interactions between system components and agents (b) prompt messages to agents, expected agent responses, if any, and rules for interpretation of agent responses (c) invocations of semantic-enabled or knowledgebase services, like semantic queries and assertions (d) descriptions of application services using service and operation names or class and method names and optional arguments (e) variable names that are to be replaced with their values at run-time (f) conditions based on run-time variable values and results of semantic queries (g) the order of execution of scenarios and scenario acts (h) aliases, or multiple ways of expressing the same meaning which are expected in inputs from interacting agents (i) translation policies related to input expressed with business domain terms (j) descriptions of events and event handling rules (k) instructions for presentation components (l) descriptions of knowledge and service resource, like application scenarios, sharing over distributed networks

Claim 4 (Previously presented). Knowledge-driven architecture system comprising: (i) semantic-enabled rules engine component or

knowledgebase, further called knowledgebase, containing business domain ontology and business rules and application scenarios that reflect application requirements (ii) Application Scenario Player capable of transforming acts of scenarios and business rules into interactions with knowledgebase, presentation components, and the underlying application services (iii) Service Connector (iv) Presenter and (v) Service components, where:

- the Application Scenario Player is connected via the Service Connector to the knowledgebase and application services and actively uses the knowledgebase in the process of application scenario interpretation;
- the Application Scenario Player interacts with the Service Connector that provides access to the knowledgebase and traditional services, as well as to the Presenter, which transforms resulting data into a proper presentation format;
- the Presenter receives data and presentation instructions from the Service Connector and interacts with multiple agents, providing one or more presentations options, like video, audio, or electronic formats, wherein said presentation instructions may include definition-rules expressed by subject matter experts with business domain terms;
- wherein the Presenter can include but not limited to (a) Formatter that prepares data for audio or video interaction or for communication to other programs and passes data further to (b) Performer that uses formatted

data for actual presentation to one or more agents via voice or screen or electronic formats for different types of agent devices

Claim 5 (Previously presented). Knowledge-driven architecture of claim 4 further comprising: special input transformation components, like speech, handwriting, or image recognition; wherein said input transformation components are connected and interact to the knowledgebase component filled with expected patterns and recognition scenarios and rules which provide transformation of multiple input types into traditional text and numeric variable values expected by event handling scenarios

Claim 6 (Previously presented). Knowledge-driven architecture of claim 4 wherein the knowledge component further comprises a service adapter to the knowledgebase, providing a standard service interface required by the Service Connector, which interacts with the knowledgebase as with a set of services

Claim 7 (Previously presented). Knowledge-driven architecture of claim 4 wherein the Service Connector component can include but is not limited to: (a) Object Retrieval that is able to find an existing service object or load the requested service class and instantiate the object at run-time (b) Object Registry that associates service objects with service and object names, stores service objects, and makes them reusable (c) Method Retrieval that retrieves the proper service method belonging to a selected service object based on the provided method arguments (d) Method Performer that performs the requested service operation on the selected service object

Claim 8 (Previously presented). Knowledge-driven architecture of claim 4 further comprising the Optimizer component wherein the Optimizer takes a snapshot of existing rules and scenarios and translates them into source code in a language such as Java or C#, which can later be compiled into binary code to fix the current application rules into a regular application that lacks flexibility but provides better performance.

Claim 9 (Previously presented). Knowledge-driven architecture of claim 4 wherein the Scenario Player contains but is not limited to: (a) Input Type Checker that checks current input and, depending on its type, submits the input to one of several interpreters (b) One or more interpreters, such as the Scenario Act Interpreter, the Prompt Response Interpreter, the New Agent Request Interpreter, etc. that, based on scenario and knowledgebase rules, translate acts of scenarios or any other input into a direct action by one of the system components (c) Queue of Scenarios that stores the current scenario when it cannot be executed at the current time, but needs to be executed later (d) Success Analysis component that can store and retrieve a history of interpretation successes and failures, and in the case of interpretation failure invokes one of several learning scenarios that prompt an agent (a user or a program) to re-define the input or to provide more details for a better interpretation

Claim 10 (Previously presented). Knowledge-driven architecture of claim 4 wherein service components are endowed with usage and value properties

Claim 11 (Original). Knowledge-driven architecture of claim 9 wherein the Success Analysis component maintains and consistently refines a list of previously used services with their APIs, keywords, descriptions, and related scenarios in the knowledgebase, and re-evaluates the usage and value properties of the services.

Claim 12 (Original). Knowledge-driven architecture of claim 9 wherein the New Agent Request interpreter uses the list of previously used services with their APIs, keywords, descriptions, and related scenarios for automatic translation of user requests into service APIs and scenario acts.

Claim 13 (Original). Knowledge-driven architecture of claim 9 wherein the New Agent Request interpreter uses a list of previously used services with their APIs, keywords, descriptions, and related scenarios to offer selected parts of this information to the user for semi-automatic translation of new requests into service APIs and scenario acts.

Claim 14 (Previously presented). Knowledge-driven architecture of claim 4 further comprising the Communicator component wherein the Communicator provides collaborative access to knowledge and services existing on other distributed network systems built with this architecture.

Claim 15 (Original). Knowledge-driven architecture of claim 11 wherein the Success Analysis component propagates via the Communicator to distributed network systems information on a new service API or a new knowledge subject after the first success operation that included the service or the knowledge subject and then after each update provided locally by the Success Analysis component.

Claim 16 (Original). Knowledge-driven architecture of claim 15, wherein information on new elements is propagated after the first successful operation that included the new element, and thereafter after each local update of such information by the Success Analysis component.